Micro Services:

Microservices are the small services that work together.

Microservices are small and independent units and runs on its own process and databse.

Microservices Principles:

================================

Single Responsibility:

The single responsibility principle states that a class or a module in a program should have only one responsibility. Any microservice cannot serve more than one responsibility, at a time.

Modeled around business domain:

Microservice never restrict itself from accepting appropriate technology stack or database. The stack or database is most suitable for solving the business purpose.

Isolated Failure:

The large application can remain mostly unaffected by the failure of a single module. It is possible that a service can fail at any time. So, it is important to detect failure quickly, if possible, automatically restore failure.

Infrastructure Automation:

The infrastructure automation is the process of scripting environments. With the help of scripting environment, we can apply the same configuration to a single node or thousands of nodes. It is also known as configuration management, scripted infrastructures, and system configuration management.

Deploy independently:

Microservices are platform agnostic. It means we can design and deploy them independently without affecting the other services.

Advantages of Microservices

=================================

1. Microservices are self-contained, independent deployment module.

2. Microservices are independently manageable services. It can enable more and more services as the need arises. It minimizes the impact on existing service.

3. It is possible to change or upgrade each service individually rather than upgrading in the entire application.

4. Faster release cycle.

5. Dynamic Scaling

6. Less dependency and easy to test.

Disadvantages of Microservices

===================================

1. There is a higher chance of failure during communication between different services.

2. Difficult to manage a large number of services.

3. The developer needs to solve the problem, such as network latency and load balancing.

4. Complex testing over a distributed environment.

Challenges of Microservices Architecture:

===

Microservice architecture is more complex than the legacy system. The microservice environment becomes more complicated because the team has to manage and support many moving parts.

Dynamic scale up and scale down:

The loads on the different microservices may be at a different instance of the type. As well as auto-scaling up your microservice should auto-scale down. It reduces the cost of the microservices. We can distribute the load dynamically.

Monitoring: :

The traditional way of monitoring will not align well with microservices because we have multiple services making up the same functionality previously supported by a single application. When an error arises in the application, finding the root cause can be challenging.

Fault Tolerance:

Fault tolerance is the individual service that does not bring down the overall system. The application can operate at a certain degree of satisfaction when the failure occurs. Without fault tolerance, a single failure in the system may cause a total breakdown.

Components of Microservices:

==============================

Spring Cloud Config Server:

Responsible for maintaining the properties/yml files specific to all services. With the help of this we can remove the duplicate properties among services.Its resolve the problem of profile specific files also. We can enable the Spring Cloud Config Server by using the annotation @EnableConfigServer.

Netflix Eureka Naming Server:

--

Eureka naming server is an application that holds information about all client service applications. Each microservice registers itself with the Eureka naming server. The naming server registers the client services with their port numbers and IP addresses.

This server will run on default port number 8761.

Eureka naming server fills the gap between the client and the middle tier load balancer.

For Example we already have two instances of CurrencyExchangeService1 and CurrencyExchangeService2 running on 8000, 8001

Suppose that we want to start another instance of currency-exchange-service that is CurrencyExchangeService3 and launch it on port 8002. Here a question arises, will ribbon be able to distribute the load to it?

The Eureka naming server comes into existence when we want to make maintenance easier. All the instances of all microservices will be register with the Eureka naming server. Whenever a new instance of a microservice comes up, it would register itself with the Eureka naming server. The registration of microservice with the naming server is called Service Registration.

Whenever a service wants to talk with another service, suppose CurrencyCalculationService wants to talk to the CurrencyExchangeService. The CurrencyCalculationService first talk with the Eureka naming server. The naming server provides the instances of CurrencyExchangeService that are currently running. The process of providing instances to other services is called Service Discovery.

Service registration and service discovery are the two important features of the naming server. In the next step, we will set up a Eureka naming server.

We need to add the below annotation to make the service as Eureka Server

```
@SpringBootApplication

@EnableEurekaServer

public class NetflixEurekaNamingServerApplication

{

public static void main(String[] args) {

SpringApplication.run(NetflixEurekaNamingServerApplication.class, args);

}

}
```

By default every thing will register under eureka. We need to add below properties to make the application as Eureka Server

```
eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false
```

Hystrix Server:

Hystrix server acts as a fault-tolerance robust system. It is used to avoid complete failure of an application. It does this by using the Circuit Breaker mechanism. If the application is running without any issue, the circuit remains closed. If there is an error encountered in the application, the Hystrix Server opens the circuit. The Hystrix server stops the further request to calling service. It provides a highly robust system.

Consider a scenario in which six microservices are communicating with each other. The microservice-5 becomes down at some point, and all the other microservices are directly or indirectly depend on it, so all other services also go down.

The solution to this problem is to use a fallback in case of failure of a microservice. This aspect of a microservice is called fault tolerance.

Fault tolerance can be achieved with the help of a circuit breaker. It is a pattern that wraps requests to external services and detects when they fail. If a failure is detected, the circuit breaker opens. All the subsequent requests immediately return an error instead of making requests to the unhealthy service. It monitors and detects the service which is down and misbehaves with other services. It rejects calls until it becomes healthy again.

Netflix Zuul API Gateway Server:

Zuul Server is an API Gateway application. It handles all the requests and performs the dynamic routing of microservice applications. It works as a front door for all the requests. It is also known as Edge Server.

Zuul is built to enable dynamic routing, monitoring, resiliency, and security.

Zuul provides a range of different types of filters that allows us to quickly and nimbly apply functionality to our edge service.

Authentication and Security: It provides authentication requirements for each resource.

Insights and Monitoring: It tracks meaningful data and statistics that give us an accurate view of production.

Dynamic Routing: It dynamically routes the requests to different backed clusters as needed.

Stress Testing: It increases the traffic to a cluster in order to test performance.

Load Shedding: It allocates capacity for each type of request and drops a request that goes over the limit.

Static Response Handling: It builds some responses directly at the edge instead of forwarding them to an internal cluster.

Netflix Ribbon:

It provides the client-side balancing algorithm. It uses a Round Robin Load Balancing.

Zipkin Distributed Server:

Zipkin is an open-source project m project. That provides a mechanism for sending, receiving, and visualization traces.

Seluth:

Responsible to maintain the unique id from request start to completion.

The Spring Cloud Sleuth token has the following components:

Application name: The name of the application that is defined in the properties file.

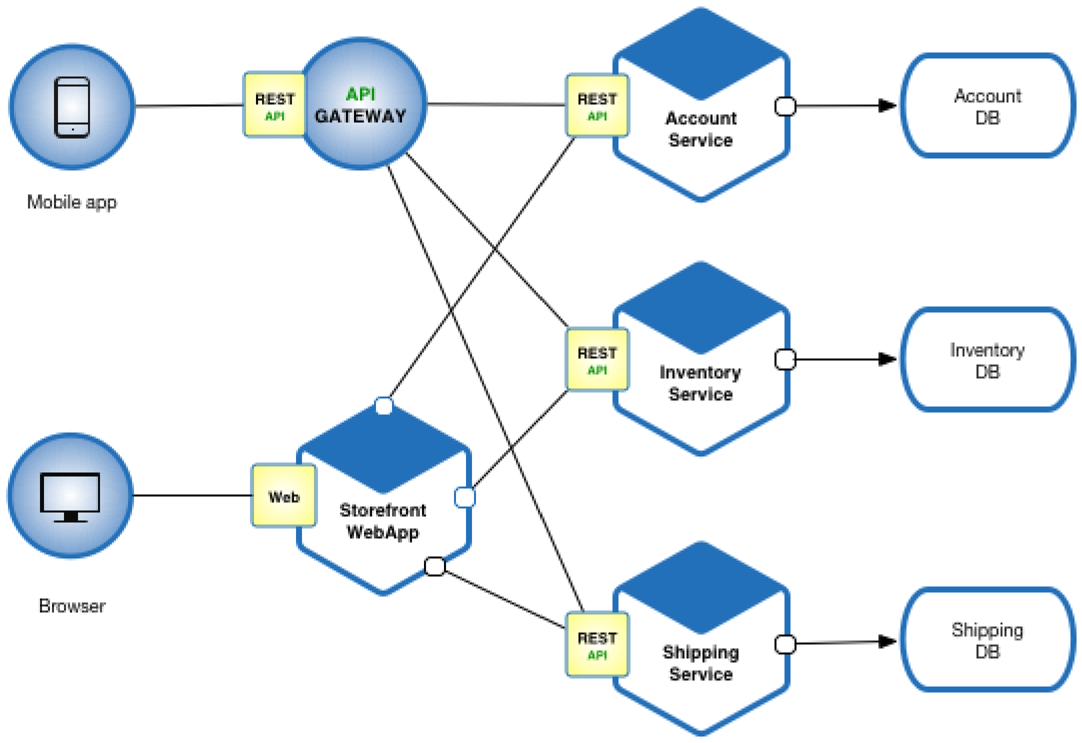
Trace Id: The Sleuth adds the Trace Id. It remains the same in all services for a given request.

Span Id: The Sleuth also adds the Span Id. It remains the same in a unit of work but different for different services for a given request.

Service and default ports:

============================

Spring Cloud Config Server	8888
Netflix Eureka Naming Server	8761
Netflix Zuul API gateway Server	8765
Zipkin distributed Tracing Server	9411


Mobile app
REST API
API GATEWAY
REST API
Account Service
Account DB
REST API
Inventory Service
Inventory DB
Browser
Web
Storefront WebApp
REST API
Shipping Service
Shipping DB